Coordinated Continual Planning Methods for Cooperating Rovers[1]

Anthony Barrett, Gregg Rabideau, Tara Estlin, Steve Chien
Jet Propulsion Lab, M/S 126-347
4800 Oak Grove Drive
Pasadena, CA 91109
818-393-5372
{firstname.lastname}@jpl.nasa.gov

Abstract—This paper compares and contrasts several coordination schemes for a system that continuously plans to control collections of rovers (or spacecraft) using collective mission goals instead of goals or command sequences for each spacecraft. A collection of self-commanding robotic systems would autonomously coordinate itself to satisfy high-level science and engineering goals in a changing partially understood environment – making the operation of tens or even a hundred spacecraft feasible.

TABLE OF CONTENTS

1. INTRODUCTION

While explicitly commanding a spacecraft via low level command sequences has worked spectacularly on previous NASA missions, there are limitations deriving from communications restrictions – scheduling time to communicate with a particular spacecraft involves competing with other projects due to the limited number of deep space network antennae. This implies that a spacecraft can spend a long time just waiting whenever a command sequence fails. This is one reason why the New Millennium program has an objective to migrate parts of mission control tasks onboard a spacecraft to reduce wait time by making spacecraft more robust [1].

In general, autonomous platforms (rovers or spacecraft), must balance long-term and short-term considerations. They must perform purposeful activities that ensure long-term science and engineering goals are achieved and ensure that they each maintain positive resource margins. This requires planning in advance to avoid a series of shortsighted decisions that can lead to failure. However, they must also respond in a timely fashion to a dynamic and partially understood environment. In terms of high-level, goal-oriented activity, the platforms must modify their collective plans in the event of fortuitous events such as detecting scientific opportunities like a sub-storm onset in Earth's magnetosphere or a Martian hydrothermal vent, and setbacks such as a spacecraft losing attitude control. For an autonomous spacecraft, the software to satisfy these requirements can be partitioned into 4 components corresponding to a belief-desire-intention (BDI) architecture [2]:

- a mission manager to generate *desires* by computing the high level science goals from commands and detected opportunities,
- a planner/scheduler to generate *intentions* by turning goals into activities while reasoning about future expected situations,
- an executive/diagnostician to generate *beliefs* by interpreting sensed events while initiating and maintaining activities, and
- a reactive controller to execute *actions* by interfacing with the spacecraft to implement an activity's primitive feedback loops.

Whether they are orbiters, probes or rovers, coordinating multiple distributed BDI agents introduces unique challenges for all four autonomy-supporting technologies. Issues arise concerning interfaces between agents, communication bandwidth, group command and control, and onboard capabilities. For example, consider a mission with a cluster of satellites simultaneously observing a point on a planet from different angles with different sensors. A certain level of communication capabilities will need to be assigned to each, possibly limiting the amount of information that can be shared between the satellites (and a ground station). The onboard capabilities also need consideration, including computing power and onboard data storage capacity. This will limit the level of autonomy each of the satellites can have. Finally, these issues apply to multiple rover missions too. A group of rovers might

[1] 0-7803-6599-2/01/$10.00 © 2001 IEEE

want to simultaneously measure vibrations caused by an explosion to determine the subsurface geology of an area on Mars.

This paper compares and contrasts 3 ways to distribute a planner/scheduler amongst a population of rovers that have separate executive/diagnosticians and reactive controllers. The first approach places the planner/scheduler on a lander that remotely commands the rovers. The second is more distributed in that it replicates a planner across the population to let each rover plan its own activities, while the lander handles goal distribution. The last approach advertises all goals and lets each rover bid for a goal based on how well its local planner can satisfy the goal given local information. These approaches delineate a space of approaches where the platform that distributes tasks maintains progressively less information on the entire population.

This paper's sections subsequently describe thought experiments for multi-rover missions that motivate 8 performance metrics for evaluating approaches toward continuous task-distribution-based coordination, explain how continual planning lets a population adapt to local conditions, compare and contrast 3 coordination methods, discuss related work, and finally conclude.

2. Multi-Platform Thought Experiments

In order to focus this discussion on distributed autonomy in space, consider different types of observations that motivate future multi-rover missions. There are 4 such kinds of observations depending on the phenomena begin measured:

- improved coverage when observing/exploring large areas (like a number of identical small rovers for exploring a remote area);
- specialized probes with explicitly separate science objectives (like a fast scout rover followed by a slower rover with more sensors);
- multi-point in-situ sensing for observing large scale phenomena that are only detectable with multiple spatially separated in-situ sensors (like a number of rovers determining chemical gradients within the Martian atmosphere); and
- building large synthetic aperture sensors with many small spatially separated sensors (like a number of rovers observing near-surface stratigraphy by making seismic tomography measurements).

These reasons for having multiple platforms in a mission are not exclusive. For instance, a rover might alternate between observing rocks in isolation and participating in a seismic tomography measurement.

Coordinating Task Distribution

In missions where each rover performs its task in isolation, the difference between an autonomous multi-rover mission and many autonomous single rover missions involves distributing tasks to the different rovers. While the task distribution for multiple autonomous single rover missions is determined on the ground, an autonomous multi-rover mission can distribute and redistribute tasks remotely. This feature improves both distribution quality and robustness by letting the population use local information to optimize the initial task distribution and to redistribute tasks when a rover suffers an anomaly, unexpectedly finishes a task early, or detects an unanticipated science opportunity.

As an example of coordinated autonomous task distribution, consider multiple rovers surveying rocks in an area on Mars using MISUS [3]. In this system a Mars lander manages a population of rovers by analyzing data from past observations, determining new observations, assigning observation goals to rovers, and collecting data as each rover moves from rock to rock, performs experiments in isolation, and analyzes its local observations (fig. 1). This system autonomously maximizes science return while minimizing the most heavily tasked rover's execution time.

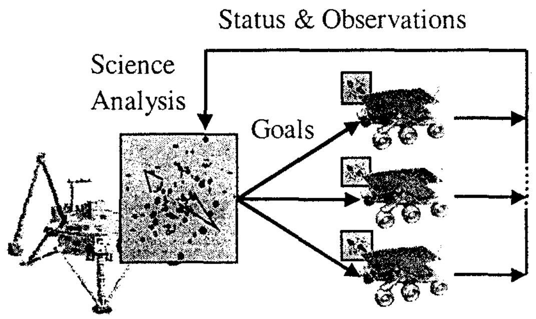

figure 1 – coordinating multiple rovers with MISUS

Coordinating Task Execution

The multi-point in-situ sensing and large synthetic aperture tasks differ operationally from the other 2 classes in that the separate rovers do not operate in isolation. For instance consider a seismic tomography measurement. To determine near-surface stratigraphy, one rover detonates an explosive charge while others measure vibrations at remote areas. By knowing the relative locations of the rovers and comparing the measured vibrations, we can determine the composition of minerals near the surface beneath the rovers. To make these measurements, the rovers must tightly coordinate their activities to assure that each rover is both correctly positioned and measuring vibration when the explosion occurs.

In many respects coordinated task execution is easier than coordinated task distribution. For the smaller missions designating a master rover that commands the other (slave) rovers as though they were physically attached solves this problem, but bandwidth restrictions keep this approach from scaling with either the number of slaves or the complexity of each slave. Resolving this scaling issue is outside this paper's focus.

Autonomy Architectures

In an earlier paper we describe 3 different autonomy architectures for a constellation of spacecraft involving leaders, followers, and slaves [4]. Here we expand this taxonomy to also include contractors. The number of autonomy modules on a spacecraft determines which of the 4 classes it falls into:

- a *slave* has no modules and is tele-operated by the reactive control module of another nearby spacecraft;
- a *follower* has both an executive/diagnostician and a reactive controller (like many existing spacecraft);
- an *contractor* has a follower's components and a planner/scheduler to optimize local activities (like DS1's remote agent experiment); and
- a *leader* has all four components.

With these 4 classes, we can define a multi-platform mission's autonomy architecture by stating the class of each platform, and how the collection of platforms coordinates its activity. In terms of MISUS, the architecture consists of having the lander lead, and letting the rovers act as followers or contractors depending on the desired local autonomy.

Given a multi-rover mission, there are two sets of metrics for evaluating the acceptability of autonomy software. The first set motivates minimizing the amount of remote autonomy and has 4 metrics:

- the amount of explicit control an operator has over the population's activities,
- the feasible accuracy of modeling the population's activities on the ground,
- the autonomy software's testability, and
- the amount of needed onboard computing power.

While the first set of metrics tend to be maximized by reducing the amount of autonomy on a constellation, the second set of 4 evaluation metrics are maximized by increasing remote autonomy:

- the population's event response time,
- the required bandwidth between platforms and to Earth,
- the quality of the downlinked data, and
- the functional redundancy.

3. CONTINUAL PLANNING

Both single and multiple autonomous rovers must respond to a (somewhat) dynamic, unpredictable environment. In terms of high-level, goal-oriented activity, a planner needs to modify rover sequences to account for fortuitous events such as observations completing early and setbacks such as a failure to traverse an assigned path.

The need to rapidly respond to unexpected events motivates continual planning, an approach where a planner continuously updates a sequence in light of changing operating context. In such an operations mode, a planner would accept and respond to activity and state updates on a one to ten second time scale. CASPER [5] is an example of a continual planner based on a heuristic iterative repair approach toward planning [6, 7]. This approach takes a complete plan at some level of abstraction and manipulates its actions to repair detected flaws. Example flaws would involve an action being too abstract to execute or many simultaneous actions with conflicting resource needs.

Making a heuristic iterative repair planner continual within a planner/scheduler module results in figure 2's algorithm. The first line assures that the PROJECTION variable always reflects how the state of a rover should evolve as its plan executes, and the fifth line causes this execution by passing near-term activities to the executive/diagnostician. Upon passing a near-term activity, a rover is committed to its execution and the planner can no longer change it – only the executive/diagnostician can. These near-term activities are defined as those that start within a domain specific amount of time in the future, and the time between now and that future point is the planner's *commit window*.

```
Given: a PLAN with multiple activities
       a PROJECTION of PLAN into the future
1. Revise PROJECTION using the currently
   perceived state and new goal activities from the
   mission manager.
2. Heuristically choose a plan flaw found in
   PROJECTION.
3. Heuristically choose a flaw repair method.
4. Use method to alter PLAN & PROJECTION.
5. Release relevant near-term activities in PLAN to
   the executive/diagnostician.
6. Go to 1.
```

figure 2 – continual planning using heuristic iterative repair

The expected future state evolution changes as a plan gets new goal activities and the perceived state diverges from expectations. This divergence is caused by unexpected exogenous events and activities having unexpected outcomes. Since a planning model can only approximate the reality experienced during execution, there is no way to guarantee the impossibility of unexpected state changes in nontrivial domains.

At any moment the projection can detect flaws in a local plan, and lines 2 through 4 select and apply repair methods to fix the flaws that appear after the commit window. For instance, a rover's observation activity can take an unexpectedly long time to complete. Depending on the delay, a subsequent observation may be impossible due to sunset occurring before the rover can reach the appropriate measurement location. A repair method might fix the flaw by rescheduling the observation to a later time, like the next morning.

Those flaws that involve committed, or executing, activities are repaired using domain specific techniques within the executive/diagnostician. Using a commit window to determine whether or not the planner/scheduler fixes a flaw is motivated by the computational cost of planning and scheduling. When a flaw appears in the commit window, a fast correct technique is needed to fix the problem, like just deleting all offensive activities. When the flaw appears after the commit window, there is time to alter the plan with slower techniques that produce more optimal results.

4. Coordinating Multiple Rovers

In an earlier paper, we compared 3 methods for coordinating a population of rovers from a central lander in the MISUS scenario described in section 2 [8]. The first and simplest method involved using a central planning to manage a population with a leader-follower architecture, where the lander generates plans that are subsequently executed by the rovers. The second method involved distributed planning where each rover planned its own activities, and the lander planned for all rovers at an abstract level to determine how to distribute goals amongst the rover population. Finally, the third method pushed all planning onto the rovers and left an auctioneer on the lander to distribute goals based on a *contract network protocol* [9, 10] – a commonly used coordination paradigm within the distributed artificial intelligence community. Within a contract net protocol, an auctioneer announces a task to a set of contractors, each contractor bids for it, and the auctioneer awards the task to the contractor with the best bid.

Within these 3 methods there was an underlying assumption that the population operated it a static well-understood environment. The system planned all activities, executed them, and then planned for the next set of activities. There was little thought about what happened when the environment became dynamic and partially understood – the motivations for continual planning.

Central Planning

The simplest way to extend continual planning for single-rover autonomy to autonomous multi-rover missions involves using a master/slave approach where a single leader performs all autonomy reasoning. The slaves only transmit sensor values to the leader and forward control signals received from the leader's reactive controller to their appropriate local devices. In this way the entire system is treated as a single multi-armed lander.

Altering this system by closing reaction loops on board the slaves to reduce the massive communication requirements results in a leader/follower approach (fig 3), where a lander uses a central planner to manage three rovers. Within the planner box, planned activities are represented as horizontal bars with effects on resources appearing below. While the planner can move some of these activities, others are fixed to represent exogenous events like sunset. The commit window overlaps the planner box and moves to the right over time. Whenever the window moves over a primitive activity, the activity is committed and sent to the appropriate executive/diagnostician. As a rover performs activities, it observes local conditions and sends state updates back to the lander to facilitate projection revision.

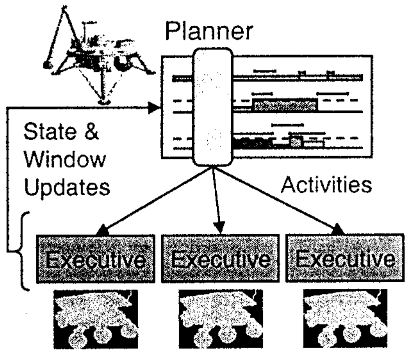

figure 3 – coordinating rovers with a central planner

With respect to our evaluation metrics, using a continual planner with a master/slave approach toward multi-platform coordination facilitates allowing a variable amount of remote autonomy. At the least autonomous extreme the continuous planner is given low-level command sequences and can only apply a go-to-safe-mode repair method upon detecting a flaw. This extreme maximizes the first set of metrics. The most autonomous extreme reduces the first set of metrics while improving the second set. Here the planner is only given a set of abstract activities and uses local information and heuristics improve event response time and the quality of downlinked data. While functional redundancy and inter-platform bandwidth are unaffected by moving from one extreme to another, turning the slaves into followers increases redundancy and reduces bandwidth. Due to how easily this change can degrade the event response time, turning slaves into followers is an active research topic in the multi-agent research community [11].

Distributed Planning

Turning followers into contractors raises issues regarding how to coordinate multiple planners. In our distributed planning approach, this coordination is achieved through using a continual goal distribution planner on the lander, and this planner continuously manages the distribution of goals based on continually updated partial information from the rovers. For instance, the architecture for a 3 rover scenario would look like figure 4, where the distribution planner might model rovers in a multi-rover scenario as points on a graph where each rover can travel a known distance from one goal activity's observation target to another's. As abstract activities fall into the distribution planner's commit window, the lander transmits them to the appropriate rovers for subsequent rover planning and execution. This use of the lander's commit window results in a tradeoff involving the window's size. A longer commit window results in letting each rover seeing more of its future activities at any given time, and this lets each rover shuffle its activities to optimize local concerns. Alternatively, a shorter commit window lets the lander delay sending a goal to a rover in order to optimize global concerns.

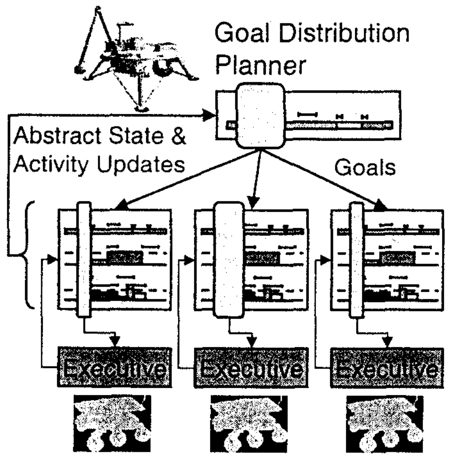

figure 4 – continual goal distribution planning

In our rover as a point on a plane characterization, the distribution-planning problem becomes a Multiple-Traveling Salesman Problem (MTSP) [12] where the members of a sales team must collectively visit each of a set of cities and the maximum traveling time of the salesmen is minimized. While this is a NP-Complete problem, there are fast greedy approaches that find slightly sub-optimal solutions. By encoding one of these approaches into our distribution planner, the lander can both determine how to distribute the goal activities and provide a rough estimate on the order in which a rover should visit its targets to perform the goal activities.

With respect to our evaluation metrics, distributed planning facilitates variable autonomy both with the ground and across the platforms. Minimizing autonomy across platforms involves making the distribution planner use full information and generate low-level action sequences for the other platforms, which can only execute their actions. This restriction turns distributed planning into the previously evaluated central planning approach.

Maximizing autonomy on the contractor platforms has the same effects as maximizing autonomy for the central planner, but also adds a reduction to inter-platform bandwidth needs. The lead platform no longer needs to maintain full state information, and each platform's planner can locally respond to events without informing the leader. Now a contractor can resolve a flaw by either quietly shuffling its local activities or reporting failure to the leader upon deleting a local activity. This quiet shuffle reduces bandwidth needs while failure reporting facilitates moving activities between platforms via the leader's continuously repairing its goal distribution plan.

Contract Networks

One way to minimize the amount of continuously updated rover information on the lander results in taking a contract network approach toward coordinating multiple planners (fig 5). Here a leader advertises each goal and each contractor bids on the goals based on its local information. To respond to an unexpected event, a contractor will either quietly shuffle its activities or delete a local activity and report failure to the leader. Upon hearing of a failure, the leader can re-advertise the failed goal for auction. Notice that there is no need for continuously updated partial contractor information – the leader does not need to know anything about the contractors to auction a goal.

Using a contract net protocol to implement a greedy solution to the MTSP involves making the lander take goal activities and incrementally advertise them to all rovers. Upon receiving a task, a rover uses its onboard planner to try to fit a solution to the goal activity into its current schedule. Upon succeeding, a rover bids its total projected travel distance after including the new observation. Rovers that fail to insert the task within a time limit do not participate in the auction. While receiving bids, the lander keeps the best bid and continually rejects lesser bids. When the auction's time limit is reached, the lander awards the task to the rover with the best bid. By bidding the total distance the rovers minimize the maximum rover travel distance – an MTSP solution. This use of a rover's continual planner to bid exposed another tradeoff involving the length of each rover's commit window. While a longer

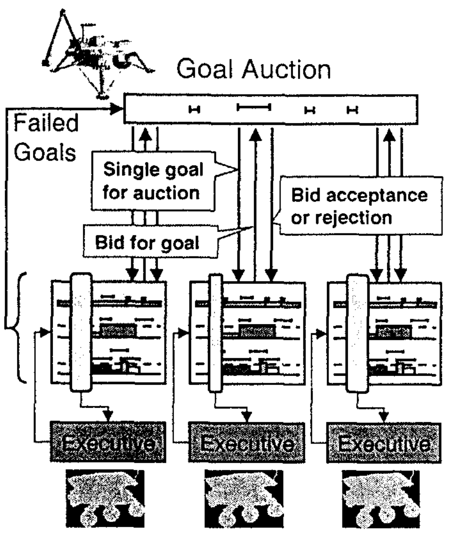

figure 5 – continual goal auction

commit window results in allowing longer auctions with more participating rovers, a shorter commit window keeps each rover from having to use the fast sub-optimal plan repair techniques.

With respect to our evaluation metrics, letting an operator restrict the platforms that can bid for certain activities results in a system with variable autonomy. At one extreme the operator can specify a low-level activity sequence for each platform, and at the other the leader gets a set of high-level goals that can go to any platform.

As before, the first extreme scores best on the autonomy minimization metrics and the second scores best on the autonomy maximization metrics. While this approach has lower inter-platform bandwidth needs than the other approaches, it has more computational overhead and assumes a greedy approach toward optimization.

5. Related Work

While there is a large literature on cooperating robots, most focuses on behavioral approaches that do not explicitly reason about partitioning goals and planning courses of action. Three notable exceptions are GRAMMPS [13], MARS [14], and RETSINA [15]. GRAMMPS is a system coordinating multiple mobile robots visiting locations in cluttered partially known environments. This system shares quite a bit similarity with our central goal allocation with distributed planning architecture for rovers. Both systems solve an MTSP problem to distribute targets, and both have low level planners on each mobile robot, but GRAMMPS focuses on path planning while learning a terrain instead of focussing on resources and exogenous events. On the other hand, MARS is a cooperative transportation scheduling system that shares many similarities with the contract net approach, and RETSINA uses peer-to-peer coordination similar to goal distribution planning. Neither MARS nor RETSINA models known exogenous events or provides default mechanisms for altering plans and transferring goals to resolve execution failures.

Other systems applying cooperating robotics research in aerospace domains include TeamAgent™ [16] and an architecture for autonomy developed at LAAS-CNRS [17]. While the TeamAgent™ taxonomy of agents participating in a population has many similarities to our slave, follower, contractor, and leader breakdown, this system focuses on a behavioral approach. The system from LAAS-CNRS on the other hand shares many similarities with our continual goal distribution planning approach. This system managed a fleet of 30 mobile robots that transported containers inside a building.

6. Conclusions

This paper compared and contrasted 3 continuous task-distribution-based coordination schemes for commanding multiple rovers with collective goals instead of goals or command sequences for each rover: central planning, distributed planning, and contract networks. All schemes supported variable autonomy and were evaluated with respect to 8 different metrics. At the lowest autonomy setting, all schemes devolved into commanding the platforms with low-level sequences, and at the highest autonomy setting the schemes differed primarily in terms of needed onboard computing, inter-platform bandwidth, and redundancy. While central planning kept all computing on the lander, distributed planning spread the computing overhead across all platforms. The result was a decrease in inter-platform bandwidth needs and an increase in redundancy with an unchanging total computing overhead. Contract networks further improved the bandwidth needs and redundancy, but this scheme also increased the total computing overhead by letting each platform see and bid for each goal.

Reasoning about incremental autonomy for distributed planning and contract networks results in a realization that these approaches toward coordinating multiple planner/ schedulers can be combined. The resultant approach would used a goal distribution planner, but would only collect enough information to limit the number of platforms that participate in an auction. One avenue for future work involves building a coordination mechanism that spans the space between contract networks and distributed planning. Another future research direction involves generating joint activities for multiple spacecraft/rovers to collectively satisfy. This would extend our approach to coordinating task execution for multiple platforms. Finally, a third research direction involves making the rovers/orbiters compete for shared resources, like communications opportunities.

ACKNOWLEDGEMENTS

This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

[1] N. Muscettola, *et al.* "On-Board Planning for New Millennium Deep Space One Autonomy," in Proceedings of IEEE Aerospace Conference 1997.

[2] A. Rao and M. Georgeff, "BDI Agents: From Theory to Practice," in Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, CA, June 1995.

[3] T. Estlin, T. Mann, A. Gray, G. Rabideau, R. Castano, S. Chien and E. Mjolsness, "An Integrated System for Multi-Rover Scientific Exploration," in *Proceedings of AAAI-99.*

[4] A. Barrett, "Autonomy Architectures for a Constellation of Spacecraft," International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS), Noordwijk, The Netherlands, June 1999.

[5] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Increase the Responsiveness of Planning and Scheduling for Autonomous Spacecraft," IJCAI99 Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World, Stockholm, Sweden, August 1999.

[6] M. Zweben, B. Daun, E. Davis, and M. Deale, "Scheduling and Rescheduling with Iterative Repair," in *Intelligent Scheduling*, Morgan Kaufman, San Francisco, 1994.

[7] A. Fukunaga, G. Rabideau, S. Chien, D. Yan 1997. "Towards an Application Framework for Automated Planning and Scheduling," In *Proceedings of the 1997 International Symposium on Artificial Intelligence, Robotics and Automation for Space*, Tokyo Japan.

[8] G. Rabideau, T. Estlin, S. Chien, A. Barrett, "A Comparison of Coordinated Planning Methods for Cooperating Rovers," AIAA 1999 Space Technology Conference, Albuquerque, NM, September 1999.

[9] G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," in *IEEE Transactions on Computers*, 29(12).

[10] T. Sandholm, "An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations," In *Proceedings of AAAI-93.*

[11] M. Tambe, "Towards Flexible Teamwork," *Journal of Artificial Intelligence Research*, 7:83-124.

[12] D. Johnson and L. McGeoch. "The Traveling Salesman Problem: A Case Study in Local Optimization," in *Local Search in Combinatorial Optimization*, edited by E. H. L. Aarts and J. K. Lenstra, John Wiley and Sons, London, pp. 215-310.

[13] B. L. Bumitt and A. Stentz. "GRAMMPS: A Generalized Mission Planner for Multiple Mobile Robots In Unstructured Environments," In *Proceedings of ICRA-98.*

[14] K. Fischer, J. Müller, M. Pischel, and D. Schier, "A Model For Cooperative Transportation Scheduling," in *Proceedings of the First International Conference on Multi-Agent Systems.* San Francisco, CA.

[15] M. Paolucci, D. Kalp, A. Pannu, O. Shehory, and K. Sycara, "A Planning Component for RETSINA Agents," Lecture Notes in Artificial Intelligence, Intelligent Agents VI. M. Wooldridge and Y. Lesperance (Eds.), forthcoming.

[16] T. Schetter, M. Campbell, D. Surka, "Multiple Agent-Based Autonomy for Satellite Constell-ations," 2000 ASA/MA Conference, September, 2000.

[17] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *Special Issue of the International Journal of Robotics Research on Integrated Architectures for Robot Control and Programming*, 17(4):315-337, April 1998.

__Dr. Anthony Barrett__ is a senior member of the Artificial Intelligence Group at the Jet Propulsion Laboratory, California Institute of Technology where his R&D activities involve planning & scheduling applied to controlling constellations of spacecraft. He holds a B.S. in Physics, Computer Science, and Applied Mathematics from James Madison University and both an M.S. and PhD in Computer Science from the University of Washington. His research interests are in the areas of planning, scheduling, and multi-agent systems.

__Dr. Tara Estlin__ is a senior member of the Artificial Intelligence Group at the Jet Propulsion Laboratory in Pasadena, California where she performs research and development of planning and scheduling systems for rover automation and multi-rover coordination. Dr. Estlin has

led several JPL efforts on automated rover-command generation and planning for multiple rovers, and is team member of the JPL Long Range Science Rover team. She received a B.S. in Computer Science in 1992 from Tulane University, an M.S. in Computer Science in 1994 and a Ph.D. in Computer Science in 1997, both from the University of Texas at Austin. Her current research interests are in the areas of planning, scheduling, and multi-agent systems.

***Gregg Rabideau** is a senior member of the Artificial Intelligence Group at JPL where he has developed several automated planning systems for spacecraft and rover applications. He is the Software Lead for the ASPEN planning system, which received honorable mention for the 1999 NASA Software of the Year award. Gregg was lead for the Data-Chaser shuttle payload planner and for the planning search component of the Remote Agent. The Remote Agent was co-winner of the same 1999 NASA Software of the Year award. More recently, he has been leading group efforts in plan optimization. He holds B.S. and M.S. degrees in Computer Science at the University of Illinois.*

***Dr. Steve Chien** is Technical Group Supervisor of the Artificial Intelligence Group and Principal Computer Scientist in the Exploration Systems Autonomy Section at the Jet Propulsion Laboratory, California Institute of Technology where he leads efforts in automated planning and scheduling for space exploration. He also is the Technology Community Lead for Autonomy for the Jet Propulsion Laboratory. Dr. Chien is also an Adjunct Associate Professor with the Department of Computer Science of the University of Southern California. He holds a B.S. with Highest Honors in Computer Science, with minors in Mathematics and Economics, M.S., and Ph.D. degrees in Computer Science, all from the University of Illinois. Dr. Chien was a recipient of the 1995 Lew Allen Award for Excellence, JPLs highest award recognizing outstanding technical achievements by JPL personnel in the early years of their careers. In 1997, he received the NASA Exceptional Achievement Medal for his work in research and development of planning and scheduling systems for NASA. He is the Team Lead for the ASPEN Planning System, which received Honorable Mention in the 1999 Software of the Year Competition and was a contributor to the Remote Agent System, which was a co-winner in the same 1999 competition. In 2000, he received the NASA Exceptional Service Medal for service and leadership in research and deployment of planning and scheduling systems for NASA.*